A Maxidle

We assume that *maxidle* is set when a class is created. *Maxidle* determines the maximum size burst allowed for a class that has sent no packets in the recent time interval.

Definitions: t, g, p. Let t be the time to transmit a packet, for the most recent packet sent from a class. Let p be the fraction of the link bandwidth allocated to that class. If the actual interpacket time for packets sent back-to-back from the class is t, then the `target' interpacket time (the time between transmitting two packets of that size at the allocated rate) is t/p, and *idle* is $t\,(1 - 1/p)$. The formula for computing *avgidle* is

$$avgidle \leftarrow g\, avgidle + (1-g)\, idle.$$

The weight g would typically be $15/16$ (that is, $g = 1 - 1/RM\_POWER$, for RM_POWER set to 2^4) or $31/32$ (for RM_POWER set to 5). The weight g determines the "time constant" of the averager.

Assume that *avgidle* initially has the value $maxidle$. Then after n back-to-back packets, *avgidle* is

$$g^n maxidle + \sum_{i=0}^{n-1} g^i\,(1-g)\, idle$$

$$= g^n\, maxidle + idle\,(1 - g^n).$$

This derivation uses the fact that

$$\sum_{i=0}^{m} g^i = \frac{1 - g^{m+1}}{1-g}.$$

If $avgidle$ reaches 0 after n consecutive packets, and $avgidle$ had the value $maxidle$ at the beginning of the burst, then the maximum size burst allowed for that class is n packets. In order to allow a maximum size burst of n packets of S bytes each, $maxidle$ should be set to

$$maxidle = t_S(1/p - 1)\frac{(1-g^n)}{g^n}, \qquad (1)$$

where t_S is the packet transmission time for a packet of S bytes.

A.1 An additional constraint on maxidle

In addition to equation (1) above, *maxidle* needs to be sufficiently large to allow for the normal variation in *idle*, and therefore *avgidle*, over one round of the round-robin scheduling. This is not a problem for classes with moderate bandwidth allocations, but an additional constraint is required for classes with bandwidth allocations greater than half the link bandwidth.

Let t be the packet transmission time for a "typical" packet. When a class sends two packets back-to-back, the resulting value of *idle* is $t - t/p$, as shown earlier. However, any class allocated less than 100% of the link bandwidth will occasionally have to wait for at least one other packet to be transmitted, and in this case the resulting value of *idle* will be at least $2t - t/p$ (making the simplifying assumption for the moment that all packets are the same size). Thus *maxidle* has to be sufficiently large not to "lose" the information that a class waited for another packet to be transmitted.

Consider a class A that has just become overlimit (e.g., *avgidle* has just become negative), and has to wait for a packet from another class to be transmitted. After that transmission, class A's value for *idle* is $2t - t/p$, and this is averaged into the previous value of $avgidle = \epsilon$ as follows:

$$avgidle \leftarrow g(-\epsilon) + (1-g)idle.$$

This gives

$$avgidle \leftarrow t\left(2 - \frac{1}{p}\right)(1-g) - g\epsilon$$

$$< t(1-g).$$

Thus, to ensure that a high-bandwidth class does not "lose" information about having waiting for some other packet to be transmitted, it is sufficient that the following condition on *maxidle* be observed:

$$maxidle \geq t(1-g).$$

A.2 Maxidle with arbitrary packet sizes

Of course, packets can come in a wide range of sizes. Assume that the actual packets are $a\,S$ bytes, for some $a > 0$, with transmission times of $a\,t_S$. Then what is the maximum number of back-to-back packets of this size that could be sent, if *avgidle* is initially at the value for *maxidle* given by the equation above?

Idle will be $a\,t_S\,(1 - 1/p)$, and after $b\,n$ back-to-back packets,

$$avgidle = g^{b\,n}\, maxidle + idle\,(1 - g^{b\,n})$$

$$= g^{b\,n-n} t_S(1/p - 1)(1 - g^n)$$

$$+ a\,t_S\,(1 - 1/p)\,(1 - g^{b\,n}).$$

We would like to know the value of b when *avgidle* first becomes zero.

Solving we get

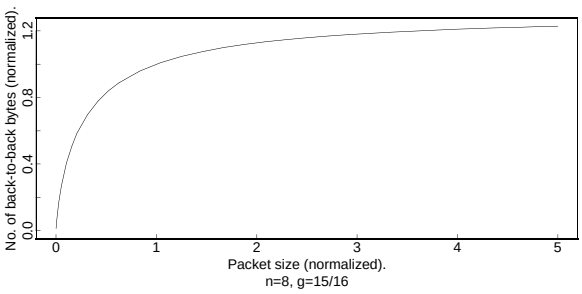
$$g^{n(b-1)} t_S(1/p-1)(1-g^n) - a\,t_S\,(1/p-1)\,(1-g^{b\,n}) = 0,$$

$$g^{n(b-1)}(1-g^n) - a\,(1-g^{b\,n}) = 0,$$

$$g^{n(b-1)}(1-g^n) + a\,g^{n(b-1)}\,g^n = a,$$

$$g^{n(b-1)}(1 + (a-1)\,g^n) = a,$$

$$g^{n(b-1)} = \frac{a}{1 + (a-1)\,g^n},$$



$g = 15/16.$

and

$$b = \frac{\log \frac{a}{1+(a-1)\,g^n}}{n \log g} + 1.$$

The initial t_S transmission time was based on S-byte packets. Now, instead of sending $n\,S$ back-to-back bytes, with packets of $a\,S$ bytes we get to send $b\,n\,a\,S$ back-to-back bytes. Figure 1 shows $a\,b$ plotted as a function of a, for $n = 8$ and $g = 15/16$. As Figure 1 shows, *maxidle* is fairly effective in controlling the maximum number of back-to-back *bytes* even for a range of packet sizes.

A.3 The calculation of avgidle

This section shows that the calculation of *avgidle* in the code in fact corresponds to the equation in [FJ95]. From [FJ95], *avgidle* is calculated from *idle* as follows:

$$avg \leftarrow (1-w)\,avg + w * diff,$$

for some weight w chosen as a negative power of two.

In the code in rm_class.c, there are two cases. When the option `USE_HRTIME' is employed, meaning that a 64-bit representation of wall-clock time is used, `avgidle` is represented in its true unscaled value, and the following equation is used:

```
avgidle +=                                (1)
((idle - avgidle) >> RM_FILTER_GAIN);
```

This is equivalent to the following:

```
avgidle +=
  (idle - avgidle) (1/2^RM_FILTER_GAIN);
```

or equivalently,

```
avgidle =
  (1 - 1/2^RM_FILTER_GAIN) avgidle
  + (1/2^RM_FILTER_GAIN) idle;
```

This gives the correct equation.

When `USE_HRTIME' is not employed, the scaled value `avgIdle` is used as follows:

```
avgIdle = avgidle * 2^RM_FILTER_GAIN,
```

for `avgidle` the true unscaled value. In this case, the following equation is used instead of equation (1):

```
avgIdle +=                                (2)
  idle - ( avgIdle >> RM_FILTER_GAIN),
```

This is therefore equivalent to

```
avgidle * 2^RM_FILTER_GAIN +=
  idle - avgidle.
```

Simplifying,

```
avgidle +=
  ( idle - avgidle )/2^RM_FILTER_GAIN.
```

Thus, equation (1) when `USE_HRTIME' is employed is equivalent to equation (2) and a scaled value for `avgIdle` when `USE_HRTIME' is not employed.

B Regulating overlimit classes: the details

Definition: undertime, now, overlimit. The CBQ scheduler checks the class variable *undertime* to see if a class can send packets without borrowing. A class is not allowed to send a packet when $undertime > now$ and the class is unable to borrow. If *avgidle* is positive after a packet has been sent from a class, then $undertime$ should be set to zero (or to something else less than the current time now).

After a packet is sent from a class, *avgidle* is updated. This section explains the equations used when *avgidle* is negative and a class that is unable to borrow therefore has to be regulated to its allocated bandwidth. If the class is to be regulated, then it must wait at least the target waiting time *ptime* before sending another packet, for

$$ptime \leftarrow t(1/p - 1).$$

If *avgidle* is negative, then the class must also wait at least

$$(1 - RM\_POWER) * avgidle$$

additional seconds, to ensure that avgidle will no longer be negative when the next packet is sent.[1]

To show that this is correct, the class will wait

$$t(1/p - 1) + (1 - RM\_POWER) * avgidle$$

seconds before sending the next packet. Assume that this packet is the same size as the last one, and also has a transmission time of t seconds. Then $idle$ will be calculated as

$$(t(1/p - 1) + (1 - RM\_POWER) * avgidle) - t(1/p - 1)$$

[1] In the simulator ns, this is called $POWEROFTWO$ instead of RM_POWER. Recall that RM_POWER is defined by the following equation, for g the weight used in the exponential weighted moving average: $g = 1 - 1/RM\_POWER$.

$$= (1 - RM\_POWER) * avgidle,$$

and the next value for $avgidle$ will be as follows:

$$avgidle \leftarrow g\ avgidle + (1-g)\ idle$$

$$= (1 - 1/RM\_POWER)\ avgidle$$

$$+(1/RM\_POWER)\ (1 - RM\_POWER) * avgidle$$

$$= 0.$$

There is an optional parameter called *extradelay_* in the ns simulator that can be used in determining how long to additionally delay an overlimit class. The parameter *extradelay_* gives the additional time interval that a overlimit class must wait before sending another packet. This parameter determines the steady-state burst size for a class when the class is running over its limit. When *extradelay_* is set to 0, then the steady-state burst size for an overlimit class is one packet. We do not discuss this further in this paper.

For the experimental code, the parameter *offtime* is used to determine how long an overlimit class is to be delayed. For a steady-state burst size of one packet, *offtime* is set to *ptime*, for *ptime* as defined in the beginning of this section. For a steady-state burst size of b packets, for $b > 1$, then *offtime* is set as follows:

$$offtime \leftarrow ptime * \left(1 + \frac{1}{(1-g)} \frac{(1-g^{b-1})}{g^{b-1}}\right),$$

for

$$g = 1 - 1/RM\_POWER$$

$$= 1 - 1/2^{RM\_FILTER\_GAIN}.$$

B.1 Controlling the minimum burstiness for a regulated class

The guidelines above for calculating undertime assume that after a regulated class sends a packet, it will have to wait the minimum possible time before sending the next packet. However, for efficiency of implementation, it might in some environments be desirable to have the regulated class wait longer after sending a packet, and to therefore send small bursts of packets, giving a steady-state burst size for the regulated class of more than one packet.

Let *offtime* be the time that the class has to wait after sending a packet. Assume that in steady state, for a class with plenty of demand that is being restricted to its link-sharing bandwidth), the CBQ implementation regulates the output for that class to a steady-state burst of n packets. (This refers to a steady-state where the class is allowed to send a burst of n packets, and then is forced to wait some time before sending another burst of n packets, and so on.) Let $avgidle_n$ be the value for $avgidle$ that allows a burst of size n before $avgidle$ reaches 0. Then

$$avgidle_n = t(1/p - 1)\frac{(1-g^n)}{g^n}.$$

Assume that a class is made to wait when *avgidle* becomes at most zero. Then we want to set *offtime* so that, if the class is allowed to send a packet after *offtime* seconds, the new value for *avgidle* will be $avgidle_{n-1}$, so that exactly $n-1$ more consecutive packets can be sent until *avgidle* reaches zero again. This is true if

$$(1-g)\ idle = avgidle_{n-1},$$

for *idle* as follows:

$$idle = offtime + t - t/p.$$

This gives

$$offtime = \frac{1}{1-g} avgidle_{n-1} + t\,(1/p - 1)$$

$$= \frac{1}{1-g}\, t\,(1/p-1)\frac{(1-g^{n-1})}{g^{n-1}} + t\,(1/p-1).$$

As examples, for $g = 15/16$, $t = 0.01$, and $n = 8$, this is

$$offtime = 0.1014(1/p - 1).$$

For $g = 31/32$,

$$offtime = 0.0896(1/p - 1).$$

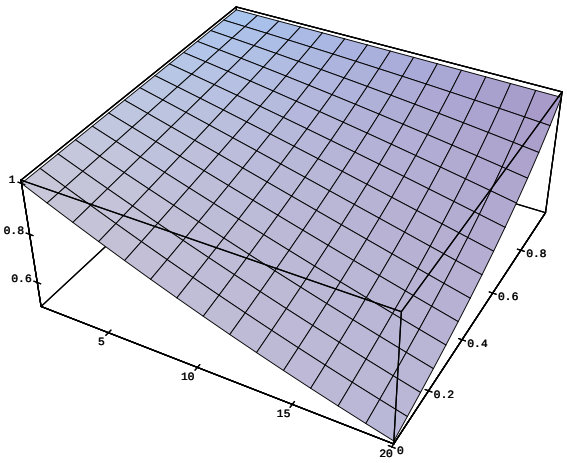
(This concurs with the findings later in this section that for a class with a steady-state burst size of n, the throughput is higher with higher values of g. As g increases, then *offtime* approaches closer to $nt(1/p-1)$, the value that would be needed for the class to achieve 100% of its throughput allocation.)

What is a class's actual throughput if this procedure is used, and a regulated class is required to send its packets in small bursts? The class transmits n packets in $n\,t$ seconds, and then waits for *offtime* seconds. Thus the actual throughput, as a fraction of the maximum bandwidth of the link, is

$$\frac{n\,t}{n\,t + offtime}$$

$$= \frac{n}{n + \frac{1}{1-g}(1/p-1)\frac{(1-g^{n-1})}{g^{n-1}} + 1/p - 1}.$$

A connection that sends bursts of n packets in this manner will get slightly less that the specified fraction p of the bandwidth, for $n > 1$. Figure 2 shows the fraction f of its allocated throughput achieved by a delayed class, for $g = 15/16$. The x-axis shows the steady-state burst size n and the y-axis shows the allocated throughput for the class. The z-axis shows the fraction of allocated throughput achieved by the delayed class. For this figure, we assume that $t = 0.01$ seconds, but the results are essentially the same for a wide range of values for t (e.g., for t as small as 0.01 ms). The results for $g = 31/32$ are similar to those in Figure 2. This data argues for a small steady-state burst size, particularly



$g = 15/16$.

for classes with small allocations. In our simulations, we generally use a steady-state burst size of $n = 1$ packet.

(What is the intuition behind this behavior? With a steady-state consisting of a burst of n packets followed by a delay, the computed $avgidle$ oscillates above and below the true steady-state average of the variable *idle*. With this mechanism, the class is delayed when the true average for idle is greater than zero, and therefore the true throughput is less than the allocated throughput.

For any $f \leq 1$, in order to guarantee that a class achieves at least the fraction f of its allocated throughput, it is sufficient to pick a steady-state burst size of at most n, for n such that

$$\frac{n}{n + \frac{1}{1-g}(1/p - 1)\frac{(1-g^{n-1})}{g^{n-1}} + (1/p - 1)} \frac{1}{p} = f.$$

Figure 3 shows the upper bound on burst size for a class to achieve at least 90% of its allocated throughput, for $g = 31/32$. Thus, for a steady-state burst size of 8 packets, a class should achieve at least 90% of its allocated throughput, regardless of the allocated bandwidth for that class.

B.2 Offtime with arbitrary packet sizes

Assume that *offtime* is pre-computed based on an assumption of a typical packet size of s bytes, with a transmission time of t_1 seconds, but that actual packets have a size of as bytes, as in Section A.2. Using the results in Section A.2, we can infer that *offtime* should be fairly effective in maintaining the steady-state burst size in bytes, even with a range of typical packet sizes. Simulations confirm that the throughput achieved by a class in bytes-per-second is fairly insensitive to the packet size in bytes.

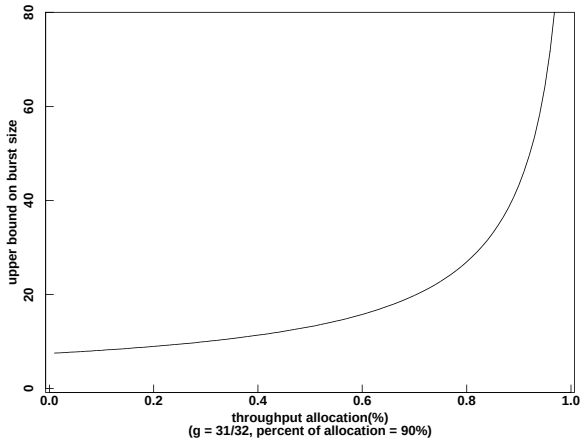


Figure 3: Upper bound on burst size for 90% of throughput, for $g = 31/32$.

References

[FJ95] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4), 1995.